Introduction to Data Science

BRIAN D'ALESSANDRO

ADJUNCT PROFESSOR, NYU

FALL 2018

FEATURE SELECTION

WHY DO FEATURE SELECTION?

The primary motivation for doing feature selection is to reduce model complexity, the benefits of which are:

- Lower expected model variance
- Easier interpretation of models
- Better scalability (both in training and deployment)

COMMON FEATURE SELECTION TECHNIQUES

- **"Naïve" methods** – pre-filter features based on heuristics
- **Best subset selection** – choose the best subset of k features from p features
- **Stepwise selection** – incrementally add/subtract features until model performance stabilizes
- **Dimensionality Reduction** – take rank-k approximations of X
- **Regularization** – Implicit, based on adding complexity penalties to loss function

NAÏVE SUBSET SELECTION

Choose top k based on a rule:

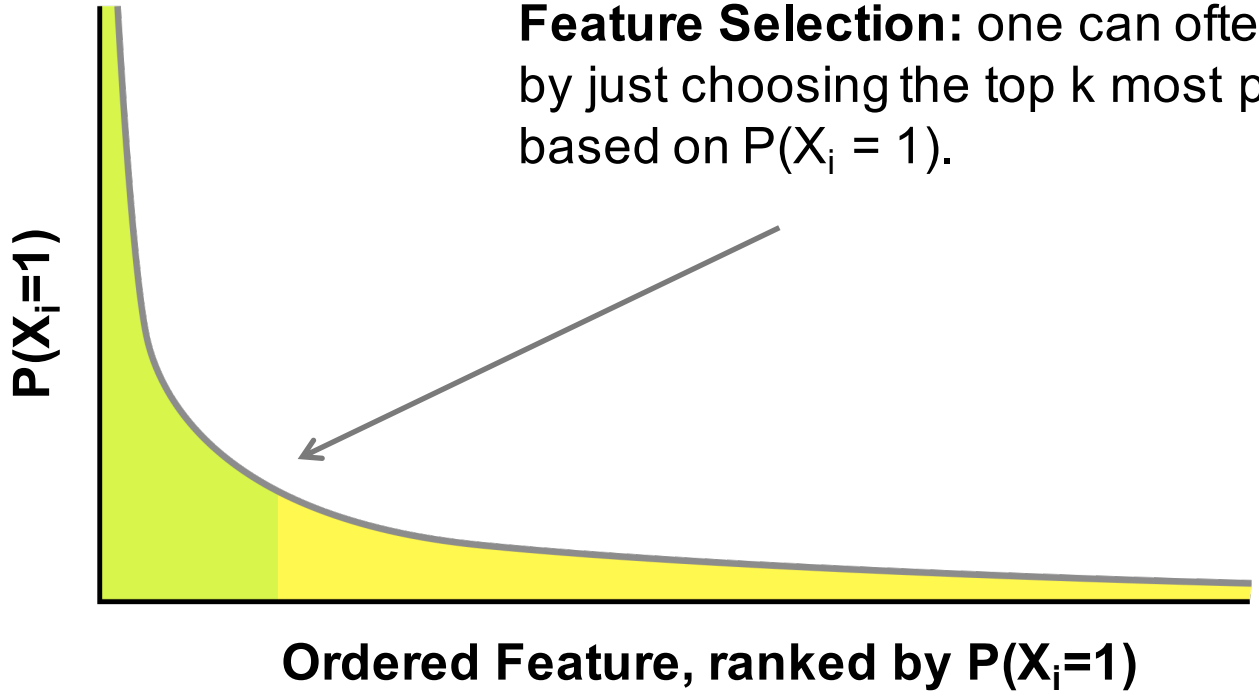
- Has highest Mutual Information/Correlation with Y
- Has the most coverage

Choosing k can be learned, but generally, the feature subset is not learned via model selection methods.

NAÏVE SUBSET SELECTION

Bag-of-words with long tail feature distributions

Some problems involve "the bag-of-words" representation. An instance can be characterized by a set of tokens U = {t_1, t_2, t_3, t_4, … t_k}. We often convert each token to a binary feature, where the token id is the feature name. This type of transformation often results in very high dimensional, but sparsely populated matrices.



Feature Selection: one can often build good models by just choosing the top k most populated feature (i.e, based on $P(X_i = 1)$.

FORWARD STEPWISE SELECTION

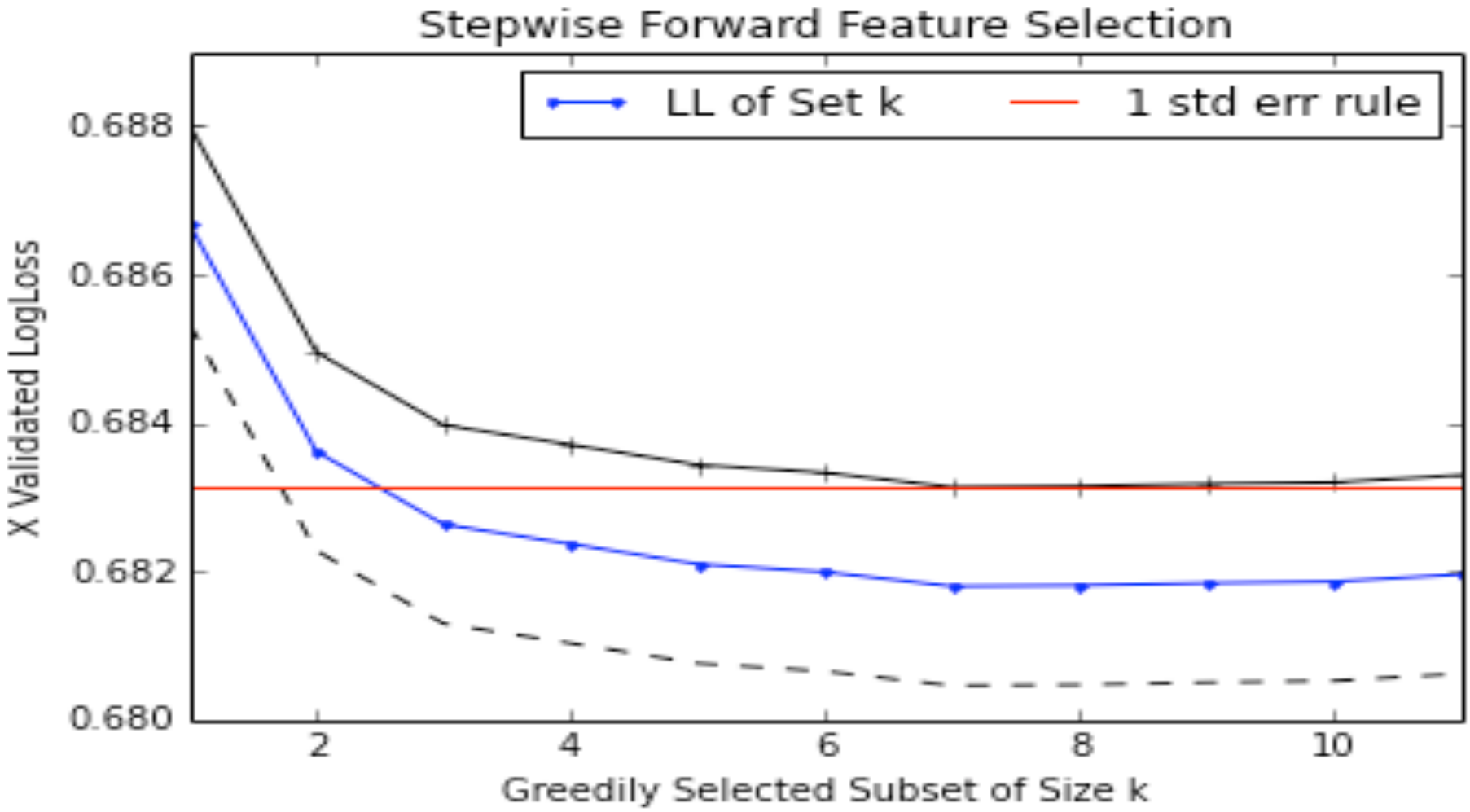
Forward stepwise selection is a greedy algorithm that incrementally selects the kth feature that improves the model after k-1 features have already been chosen.

Algorithm

- 1. Initialize: Curr_Best_Subset = { } (i.e., just the intercept)
- 2. Loop through each feature j:
  - a. Add the feature to the current best subset, train a model
  - b. Get out-of-sample error on the model trained with Curr_Best_Subset+feature j
- 3. Choose the feature with the best out-of-sample improvement in error
- 4. Add this best feature to Curr_Best_Subset, and log the out-of-sample error
- 5. Repeat steps 2 - 4 until stopping criterion is met.

FORWARD STEPWISE SELECTION

This plot shows the cross-validated LogLoss as we iteratively build the feature set, choosing the next best feature as the one that improves the cross-validated error the most. There are multiple ways to define a stopping criterion.



FEATURE REDUCTION VIA REGULARIZATION

IMPLICIT FEATURE SELECTION

Regularization can be thought of as an implicit feature selection tool.

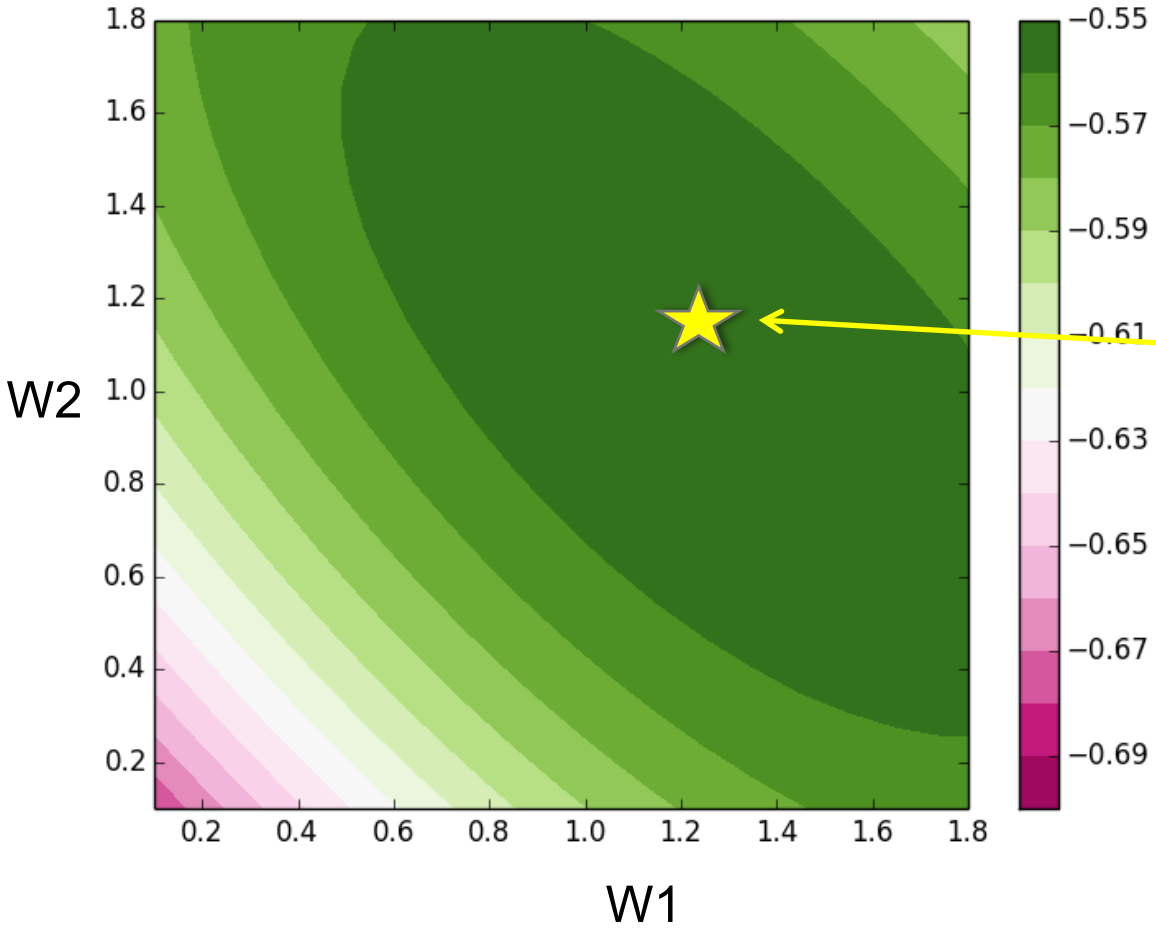
We control the complexity of the model by restricting how large the feature weights can grow.

Favors more biased models as a means to reduce expected variance.

Useful when: n is small, dimensionality is high

OPTIMIZATION REVISITED

In standard ERM we look for the feature weights that minimize our loss function. We generally don't put any limits on what values the weights can take.



Objective

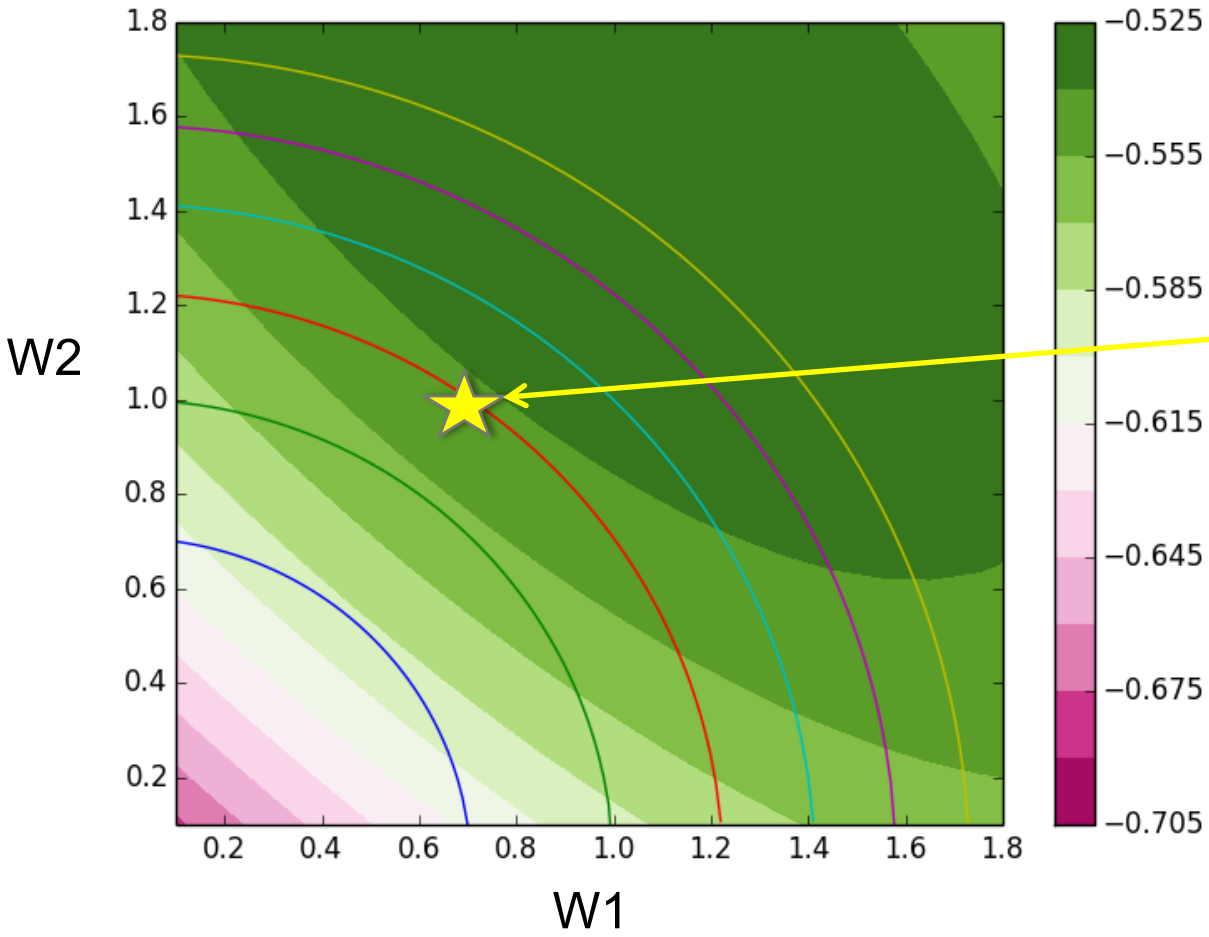
$$f^{opt} = \underset{f \in \mathbb{F}}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \mathbb{L}(f(x_i), y_i)$$

Solution

Use some convex optimization procedure to numerically solve.

CONSTRAINING THE SOLUTION

We're going to use the same objective function, but we're going to add a constraint. We specify that the norm of the weights can't grow beyond a certain value. Usually R(W) is a convex function, such as L2 or L1 norm of W.



Objective

$$f^{opt} = \underset{f \in \mathbb{F}}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \mathbb{L}(f(x_i), y_i)$$

subject to $R(W) \leq t$

Solution

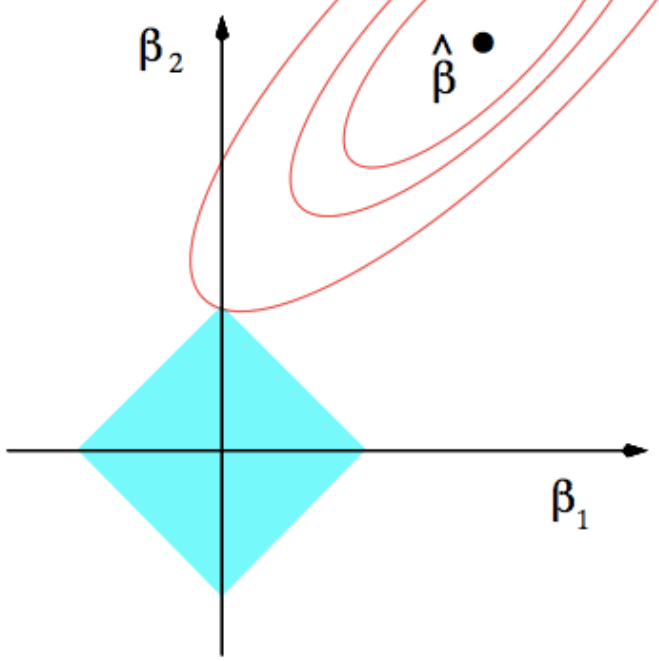
Now we pick the max value that lies within the constraint circle.

L1 AND L2 REGULARIZATION

We can take the Lagrange form of the constrained optimization problem, and set up two new objective functions.

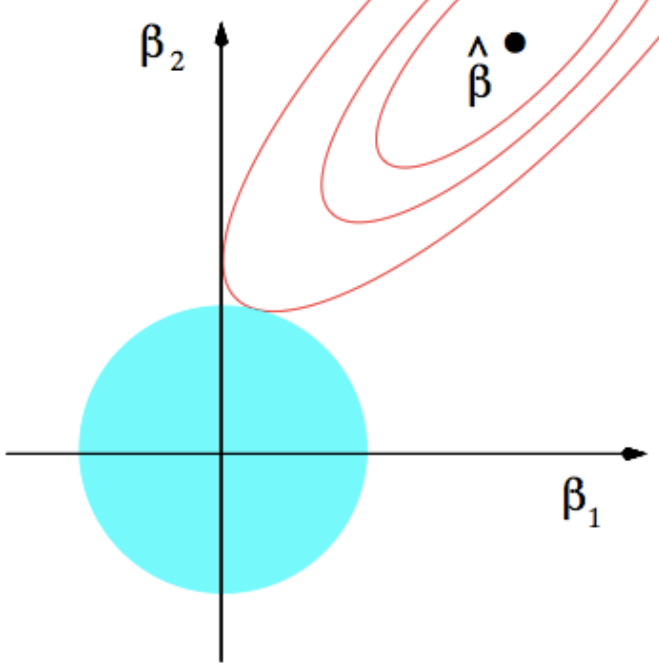
L1 (Lasso)

$$f^{opt} = \underset{f \in \mathbb{F}}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \mathbb{L}(f(x_i), y_i) + \lambda \sum_{j=1}^{m} |W_j|$$



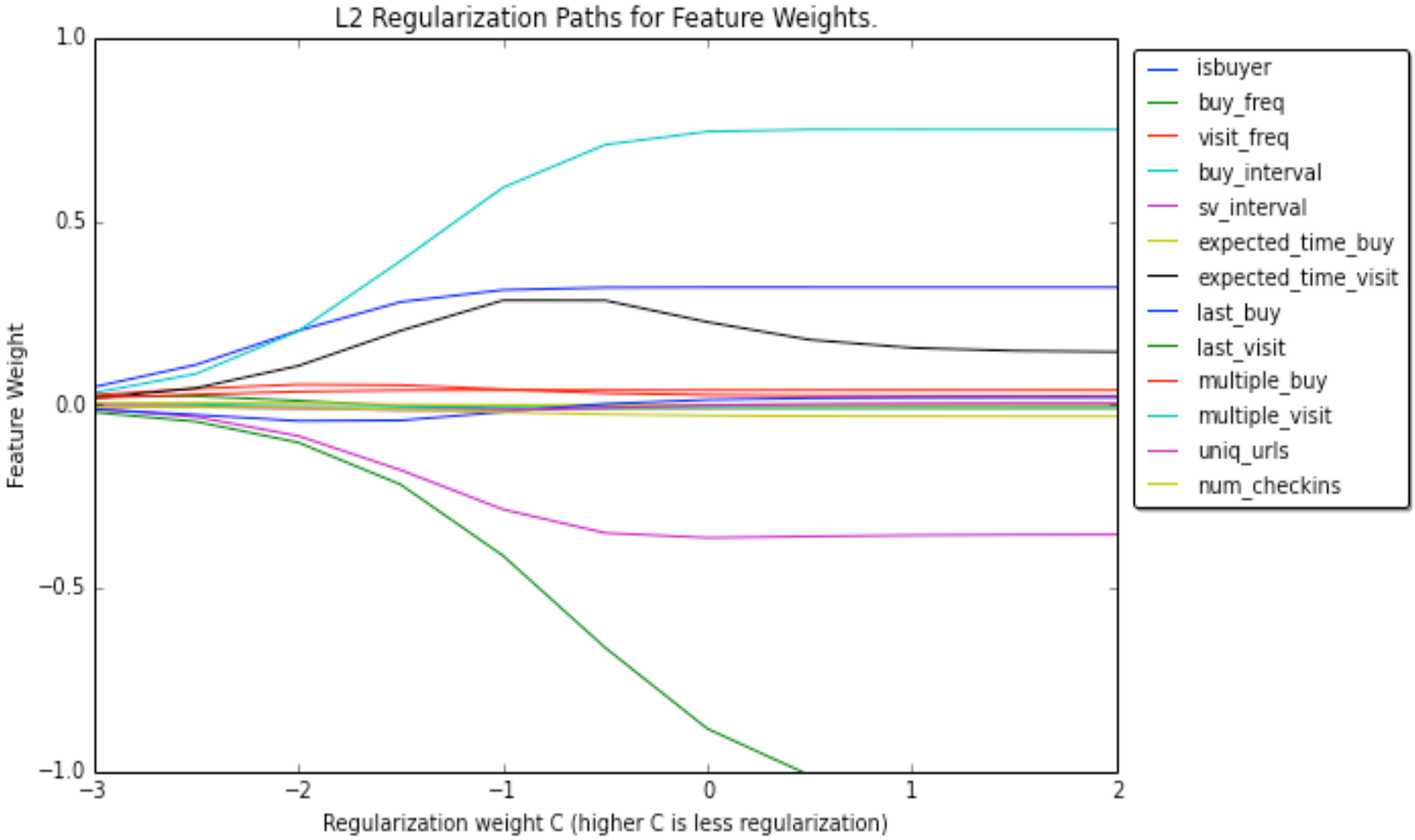
L2 (Ridge)

$$f^{opt} = \underset{f \in \mathbb{F}}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \mathbb{L}(f(x_i), y_i) + \lambda \sum_{j=1}^{m} W_j^2$$



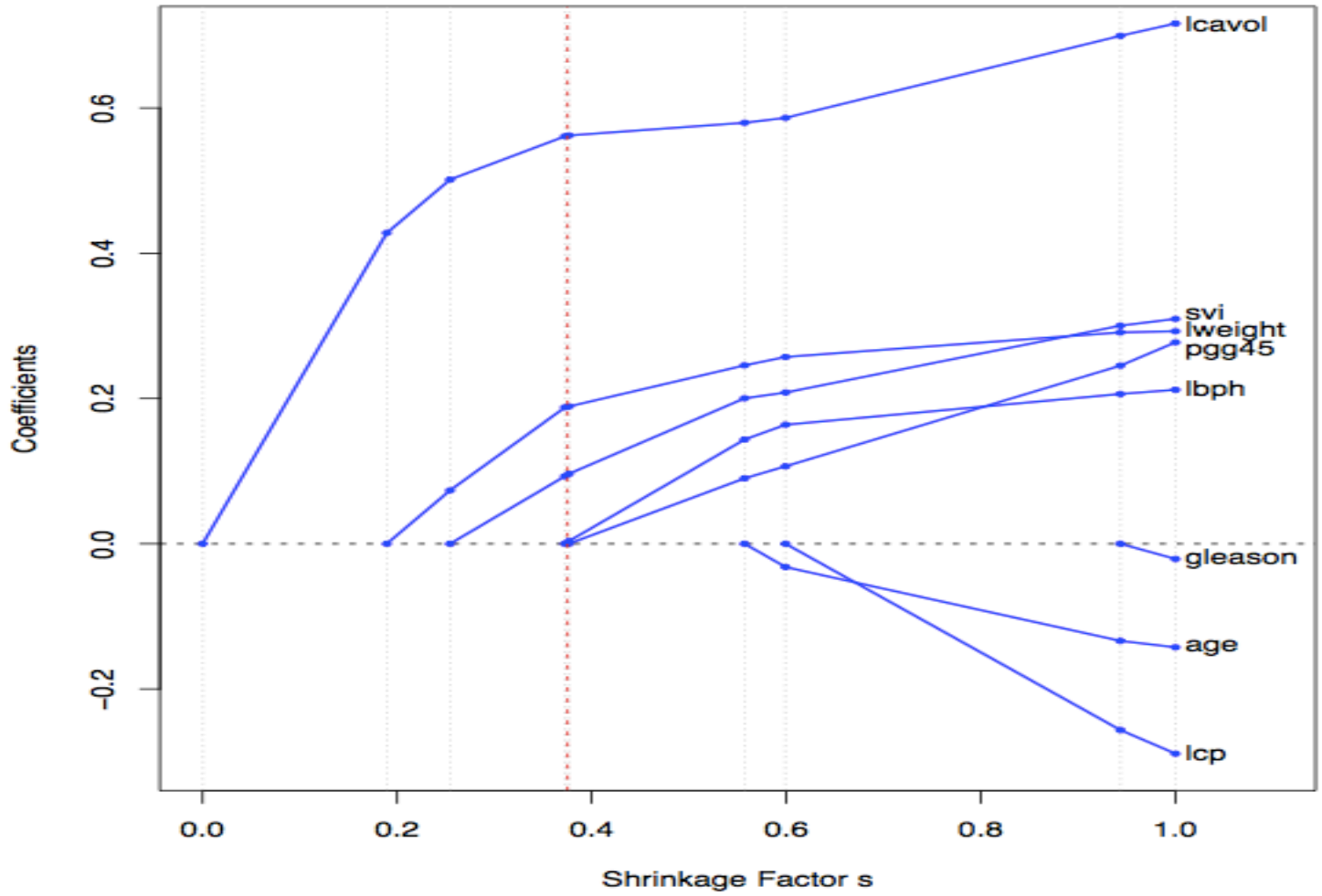
REGULARIZATION PATHS

We can visualize what happens to the feature weights as we change the regularization strength.



AN L1 EXAMPLE

This is an example from ESL, we can see here how coefficients are frozen at 0 before they grow. This is a good example of how regularization is a form of implicit feature selection.



WHY DOES IT WORK?

Regularization is a tool to reduce a model's complexity, where 'complexity' represents how sensitive a model is to small levels of noise in the data.

Take a standard linear model: $F(X) = W*X + b$

Define: $X^* = X + \varepsilon$, s.t. $|\varepsilon|_2$ is reasonably small

Since X and X^* are reasonably close, we hope that $F(X)$ and $F(X^*)$ are reasonably close. We can measure this by:

$$|F(X) - F(X^*)| = |W*X - W*X^*| = |W*(X-X^*)| = |W^* \varepsilon| <= |W|_2 * |\varepsilon|_2$$

So by bounding $|W|_2$ we can limit how much small pertubations of X changes our prediction.

In other words, regularization ensures that the nearest neighbors of X receive a similar prediction as X.

THE BAYESIAN INTERPRETATION

We can think of regularization as Maximum a Posteriori estimation problem.

Using Bayes rule, the posterior of β is:

$$P(\beta|X,Y) \propto P(X,Y|\beta) * P(\beta) = Likelihood * Prior$$

The MAP estimate of β is the value that maximizes the posterior distribution.

$$\hat{\beta}_{MAP} = \underset{\beta}{\text{argmax}}\, L(\beta|X,Y)\, P(\beta)$$

BAYESIAN LOGISTIC REGRESSION

For L2 regularization, we assume that P(β) is drawn from a normal distribution with 0 mean and variance τ.

$$P(\beta_j) = N(0, \tau_j) = \frac{1}{\sqrt{2\pi\tau_j}} exp(\frac{-\beta_j^2}{2\tau_j}), \quad j = 1, 2, \ldots d$$

The posterior of logistic regression feature weights can then be defined as:

$$\hat{\beta}_{MAP} = \underset{\beta}{\text{argmax}} \prod_{i=1}^{n} p_i^{y_i}(1 - p_i)^{1-y_i} \prod_{j=1}^{d} \frac{1}{\sqrt{2\pi\tau_j}} exp(\frac{-\beta_j^2}{2\tau_j})$$

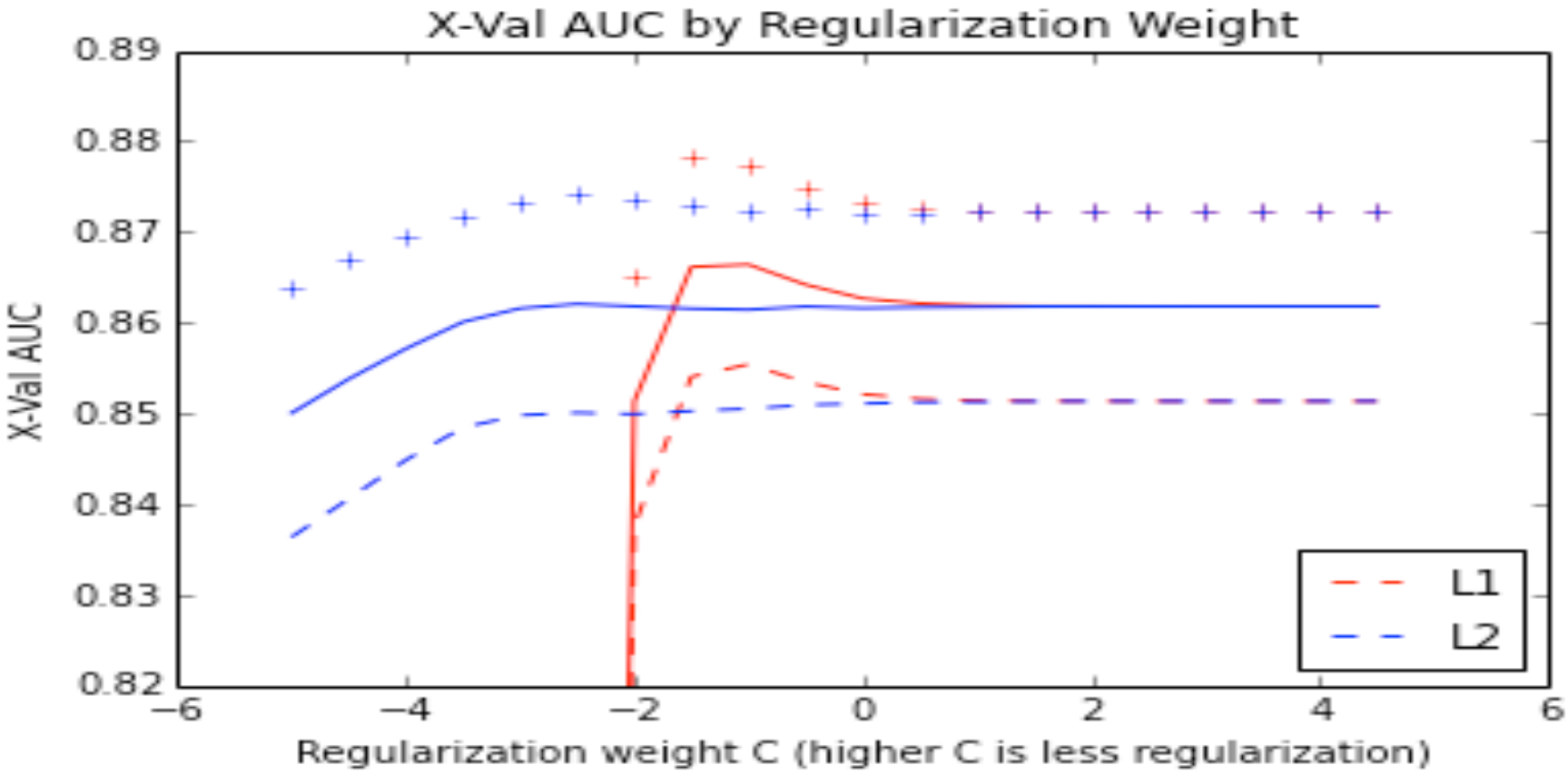
If we take the negative log of the above posterior, we end up with the ERM form of logistic regression.

(See iPython notebook for similar treatment of L1).

REGULARIZATION WEIGHT

We generally always want to use some model selection methodology (i.e., cross-validation) to choose the optimal regularization strength.

We can use the same method for L1 vs L2, but sometimes we choose L1 on principal alone. Its often desired to have the implicit feature selection.



FEATURE REDUCTION VIA DATA COMPRESSION

SINGULAR VALUE DECOMPOSITION

Although this is not a linear algebra course, this equation happens so often in data analysis that it is worth learning (again).

$$X = U \Sigma V^T$$

NxM NxM MxM $(MxM)^T$

We will not dive into all of the theoretical aspects of SVD and related topics. Our goal here is to understand it intuitively and be able to use it as a tool for solving other common Data Science problems.

SINGULAR VALUE DECOMPOSITION

Lets go through this piece by piece:

U

U holds the left singular vectors. Each row contains k elements that correspond to the latent factors of each row of X. U is orthonormal.

Σ

Σ is a diagonal matrix that holds the singular values (in descending order). The singular values are essentially weights that determine how much that latent factor contributes to the matrix.

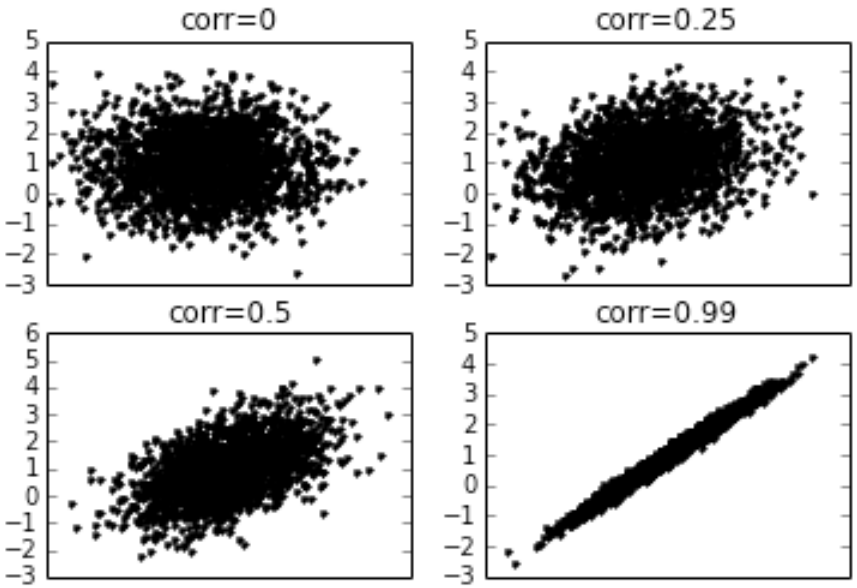
V^T

V holds the right singular vectors. Each row contains k elements that correspond to the latent factors of each column of X. V is orthonormal.

SINGULAR VALUE DECOMPOSITION

How does this relate to data structure?

Lets recall these scatter plots. Each plot can be expressed as an Nx2 Matrix. The SVD is a tool that can help us understand how much information or structure is actually in the matrix.

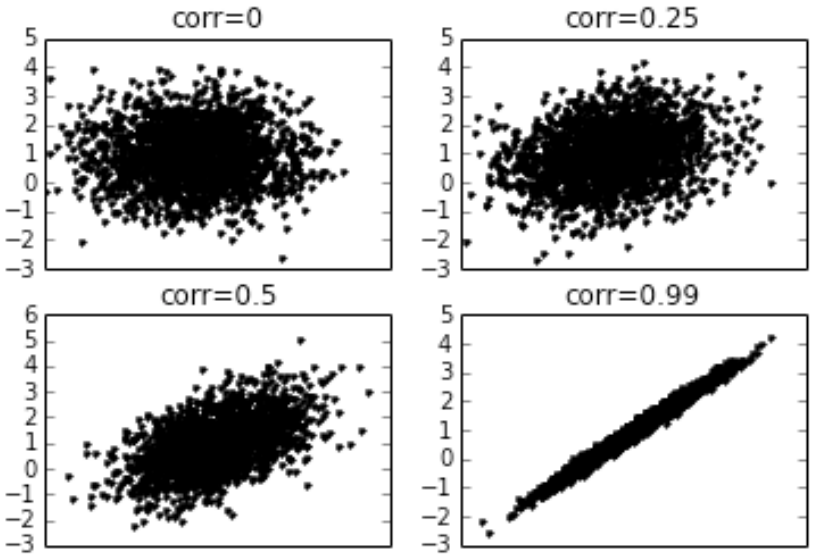


More independence of columns = more information or degrees of freedom.

Less independence of columns = less information or degrees of freedom.

SINGULAR VALUE DECOMPOSITION

Using Scipy we can easily compute the SVD.:
U,Sig,Vt = scipy.linalg.svd(matrix)



$$\Sigma = \begin{bmatrix} 45 & \\ & 44 \end{bmatrix} \qquad \Sigma = \begin{bmatrix} 50 & \\ & 38 \end{bmatrix}$$

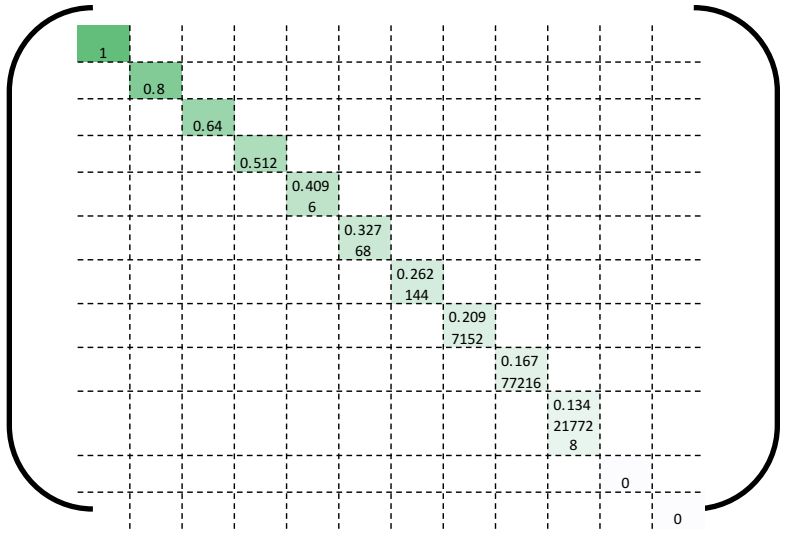
$$\Sigma = \begin{bmatrix} 53 & \\ & 32 \end{bmatrix} \qquad \Sigma = \begin{bmatrix} 62 & \\ & 5 \end{bmatrix}$$

The magnitude of the singular-values are generally determined by the magnitude of the values in X. The relative difference between singular values is a function of the level of independence of the columns.

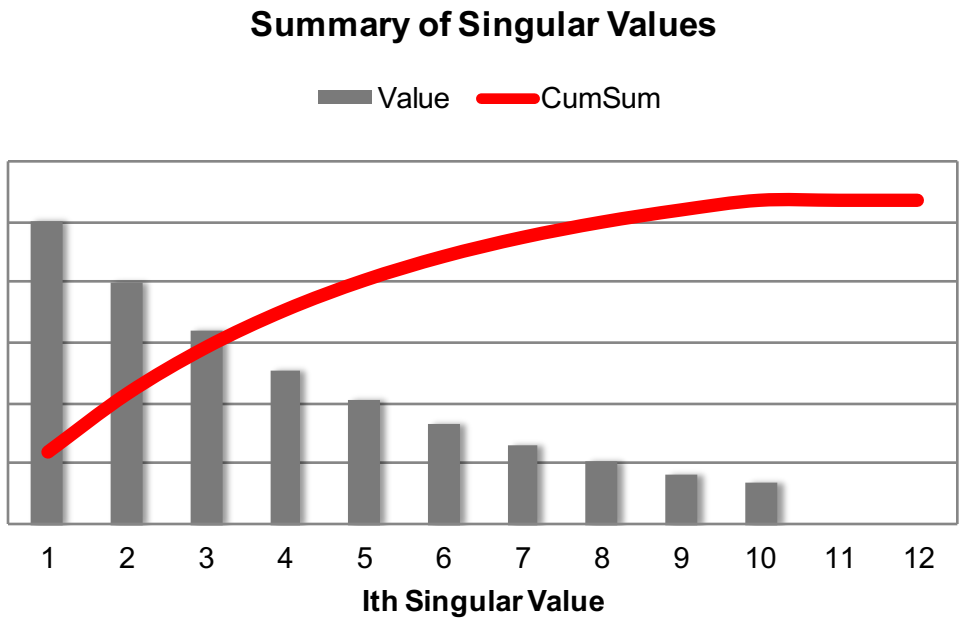
SINGULAR VALUE DECOMPOSITION

This idea generalizes to more dimensions, which is where the SVD is extremely useful.

$\Sigma =$

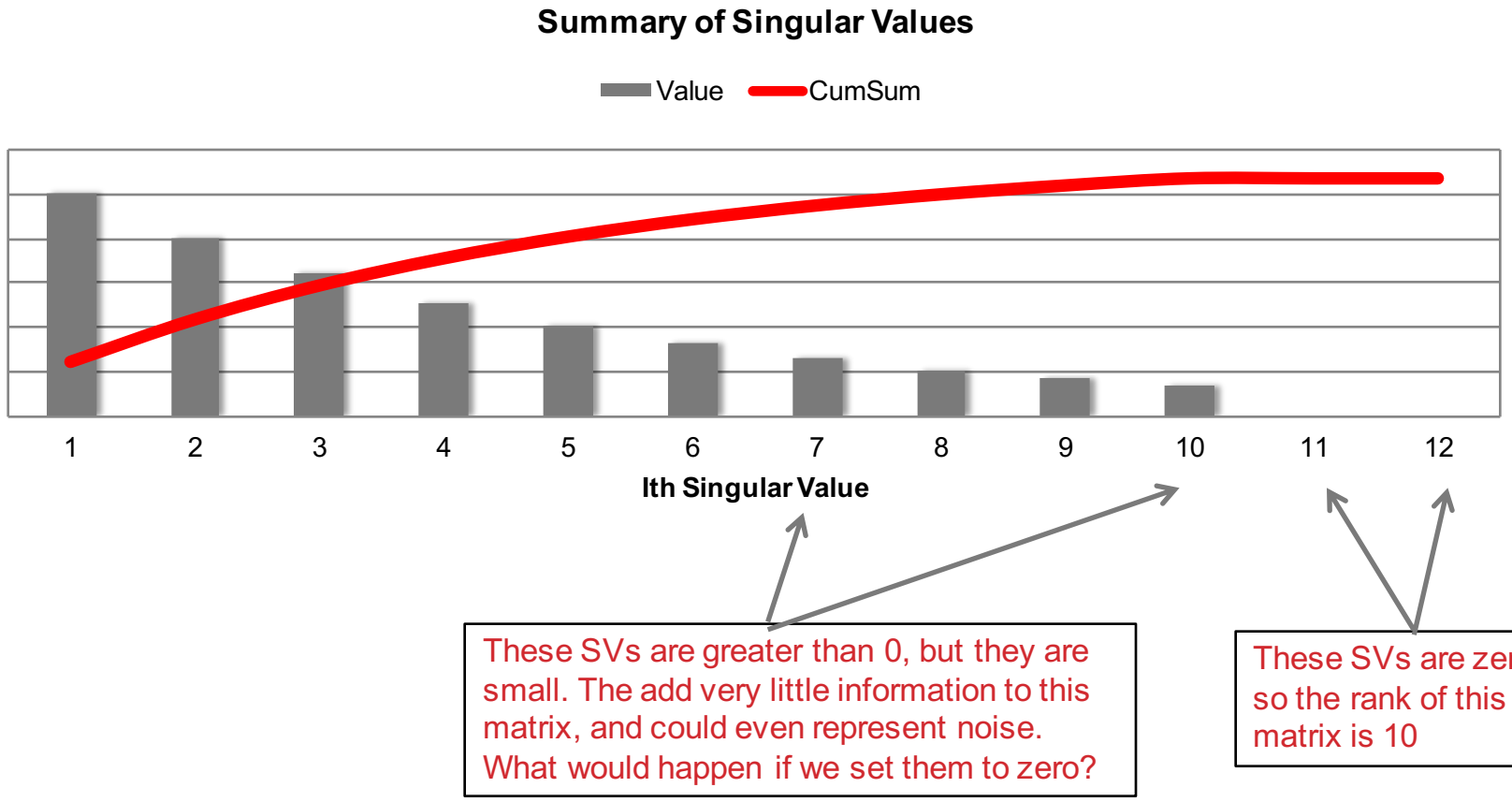


The skew of the singular values, and the shape of the cumulative sum curve can give us a sense of the degree of independence in a multi-dimensional matrix.



THE LOW RANK APPROXIMATION

The rank of a matrix is the size of the largest number of independent columns of a matrix. The rank can be found by counting the number of singular values >0.



THE LOW RANK APPROXIMATION

We can build a matrix X_k that approximates our original matrix by doing the following:

1. Compute the SVD of X
2. Truncate U, Σ and V to take only the k highest columns & singular values
3. Multiply back together to get X_k

Low Rank Approximation – K<<M

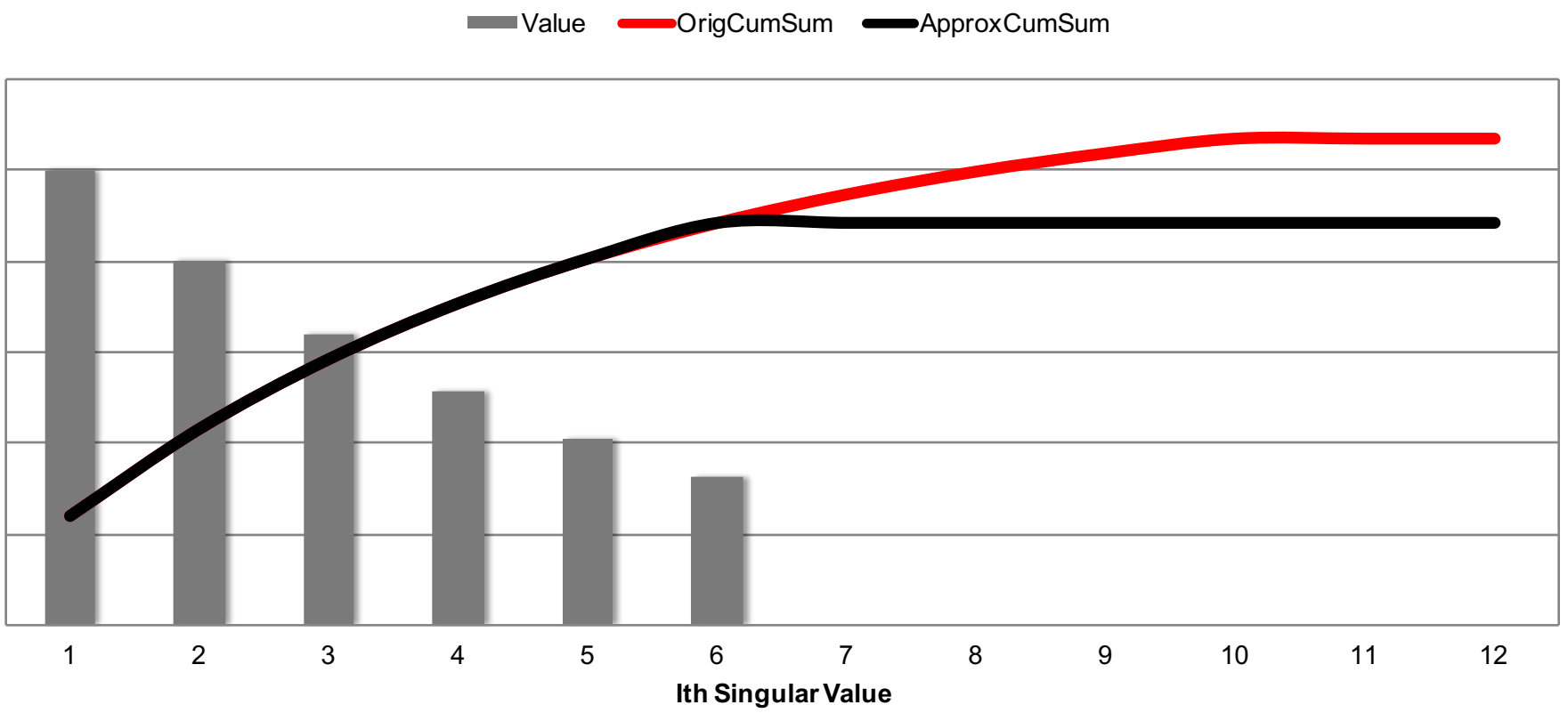
$$X_k = U_k \Sigma_k V_k^T$$

NxM *NxK* *KxK* $(MxK)^T$

THE LOW RANK APPROXIMATION

Our original matrix had 12 columns with a rank of 10. In our approximation, we decided to use k=6. The cumulative sum of singular values is related to the amount of information contained in the matrix. By using k=6, we lost some information, but not half.

What do we gain with the low rank approximation?
What do we lose?



THE COST

This result comes from the Eckart-Young-Mirsky Theorem

If X is an NxM matrix and X_k is the rank-*k* approximation derived from the SVD, then the sum-of-squares error between the entries of X and X_k equals the square-root of the sum-of-squares of the singular values > k.

i.e.

$$\|X - X_k\|_F = \sqrt{\sum_{r=k+1}^{min(m,n)} \sigma_r^2}$$

Where the Frobenius norm is defined as:

$$\|A\|_F = \sqrt{\sum_{i=1}^{n} \sum_{j=1}^{m} a_{ij}^2}$$

With this theorem we can think of the low-rank SVD as a least-squares regression against the original matrix X.

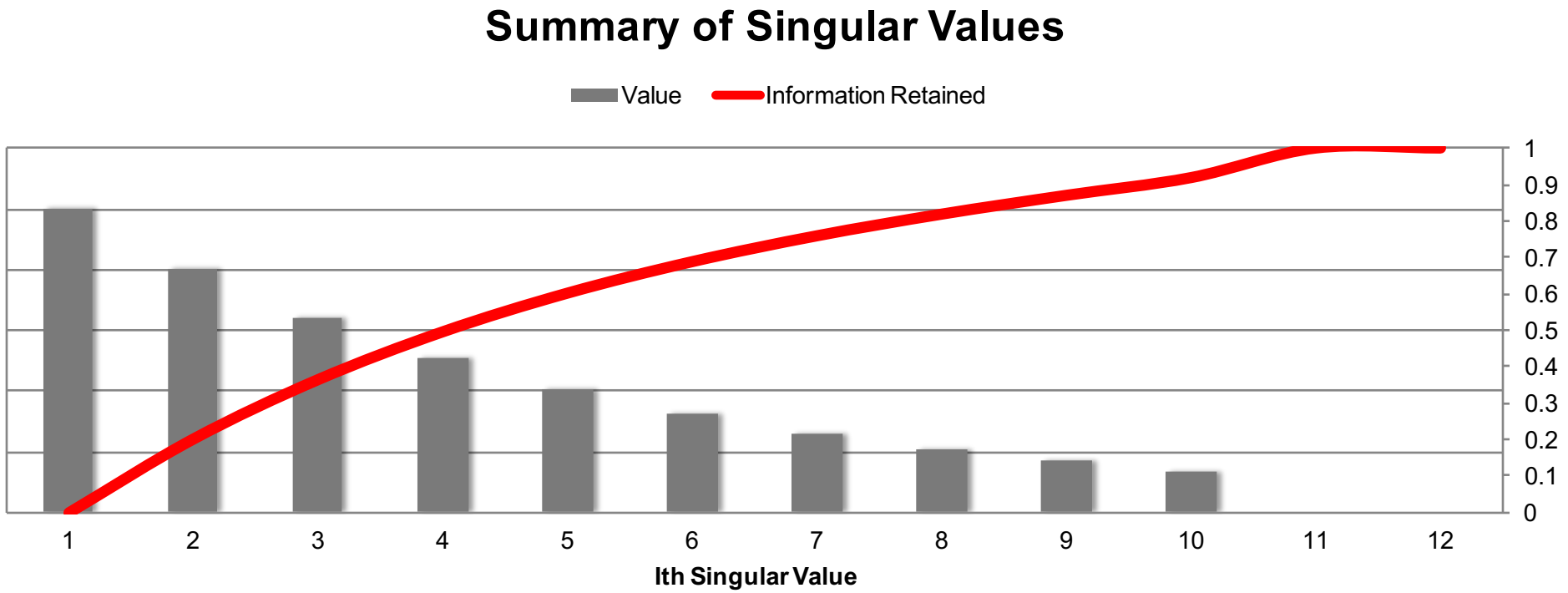
THE COST - EXAMPLE

Given a matrix with the Singular Values shown below, the "Information Retained" shows:

$$1 - \frac{\|X - X_k\|_F}{\|X\|_F}$$

This is very similar to the R^2 metric in linear regression, and it tells us how well our rank-*k* approximation "fits" the the original matrix from a least squares sense.

Analyzing a rank-*k* approximation like this gives us a tool to choose *k*.



SVD BASED FEATURE SELECTION

SVD can be used as a dimensionality reduction technique. Like subset methods, we choose k features, but the features are the transformed orthogonal columns of the rank-k approximation of X.

Compute rank-k SVD $X_k = U_k \Sigma_k V_k^T$

We can create an NxK reduced matrix with $X V_k$ This effectively becomes our new "X" matrix, and we do any analysis (clustering, modeling, etc.) with the reduced matrix.

We can project any new data into the reduced space by, $X' V_k$ and then use this in our algorithms.